Model Checking Stencil Computations Written in a Partitioned Global Address Space Language

Tatsuya Abe, Toshiyuki Maeda, and Mitsuhisa Sato
RIKEN AICS

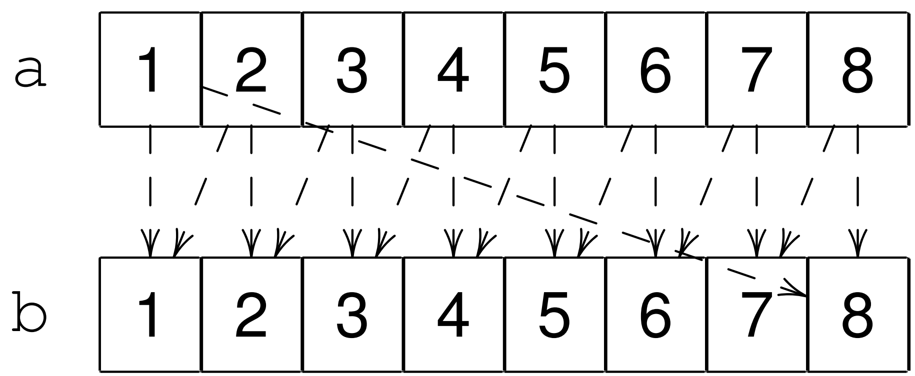
HIPS'13
May 20, 2013

What is Stencil Computation?

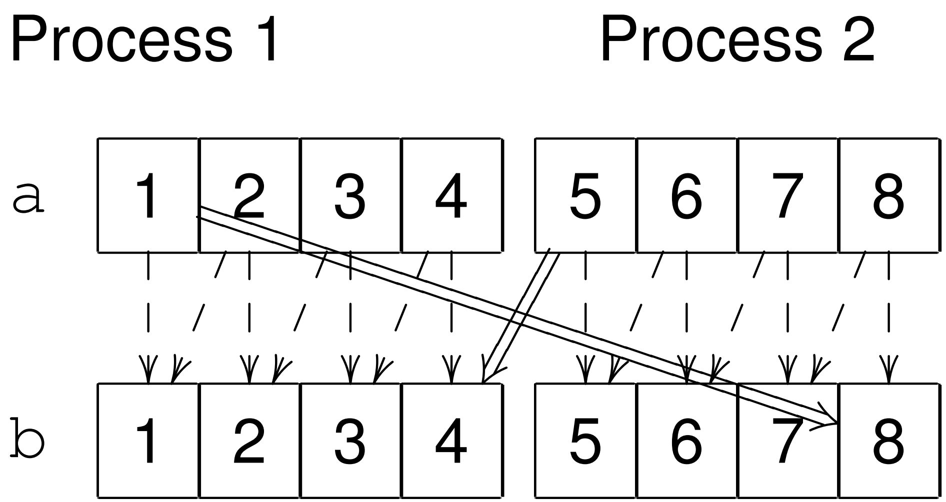
Update each array element using its neighboring elements.

Example code:

```
do i=1,8
  b(i)=a(i)+a(i+1)
end do
```



Parallelizing Stencil Computation on Multiple Nodes



Need to copy boundary elements between processes.

Stencil Computation in MPI

Code *how* to communicate among computational nodes.

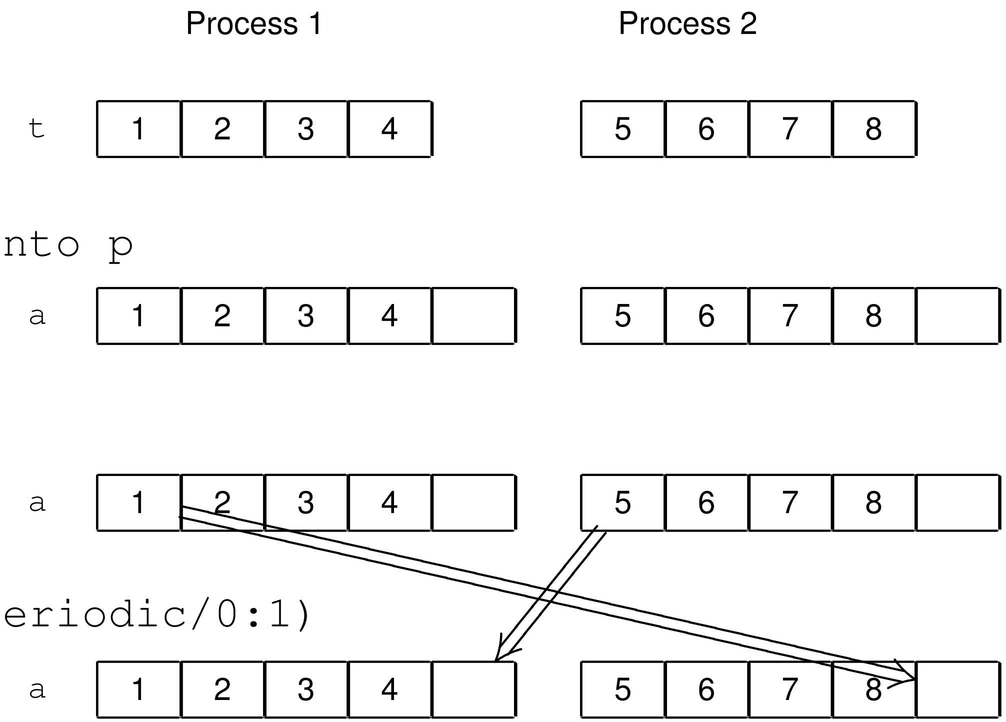
```
call MPI_COMM_RANK(MPI_COMM_WORLD, me,
  ierr)
...
you=mod(me+1,2)
call MPI_IRECV(a(5), 1,
  MPI_DOUBLE_PRECISION, you,
  MPI_ANY_TAG, MPI_COMM_WORLD, req, ierr)
call MPI_SEND(a(1), 1,
  MPI_DOUBLE_PRECISION, you,
  MPI_ANY_TAG, MPI_COMM_WORLD, ierr)
call MPI_WAIT(req, stat, ierr)
do i=1,4
  b(i)=a(i)+a(i+1)
end do
```

Example Stencil Computation in a PGAS language: XcalableMP

Code *what* we compute by using all nodes.

```
!$xmp nodes p(2)
!$xmp template t(8)
!$xmp distribute t(block) onto p
!$xmp align a(i) with t(i)
!$xmp align b(i) with t(i)
!$xmp shadow (0:1) :: a
...
...
!$xmp reflect (a) width (/periodic/0:1)

!$xmp loop on (i)
do i=1,8
  b(i)=a(i)+a(i+1)
end do
```



← the original stencil code remains unchanged

Common (Frequently Seen) Mistakes in XcalableMP

Directives are inserted inappropriately

- reflect directive is missing
- reflect directive is redundant

Missing Reflect

```
!$xmp shadow (0:1) :: a
...
!$xmp loop on (i)
do i=1,8
  a(i)=... ← a is updated
end do

!$xmp reflect (a) width (/periodic/0:1) ← missing
!$xmp loop on (i)
do i=1,8
  b(i)=a(i)+a(i+1) ← b is computed by an old value in the shadow
end do
```

Redundant Reflect

```
!$xmp shadow (0:1) :: a
...
!$xmp reflect (a) width (/periodic/0:1)   ← to compute b
!$xmp loop on (i)
do i=1,8
  b(i)=a(i)+a(i+1)
end do
```

↓ to compute c, try to reflect, but

```
!$xmp reflect (a) width (/periodic/0:1)   ← redundant
!$xmp loop on (i)
do i=1,8
  c(i)=3*a(i)+4*a(i+1)
end do
```

Our Approach to Find Bugs: Model Checking

Verify a property of a program by exploring all the states the program can reach.

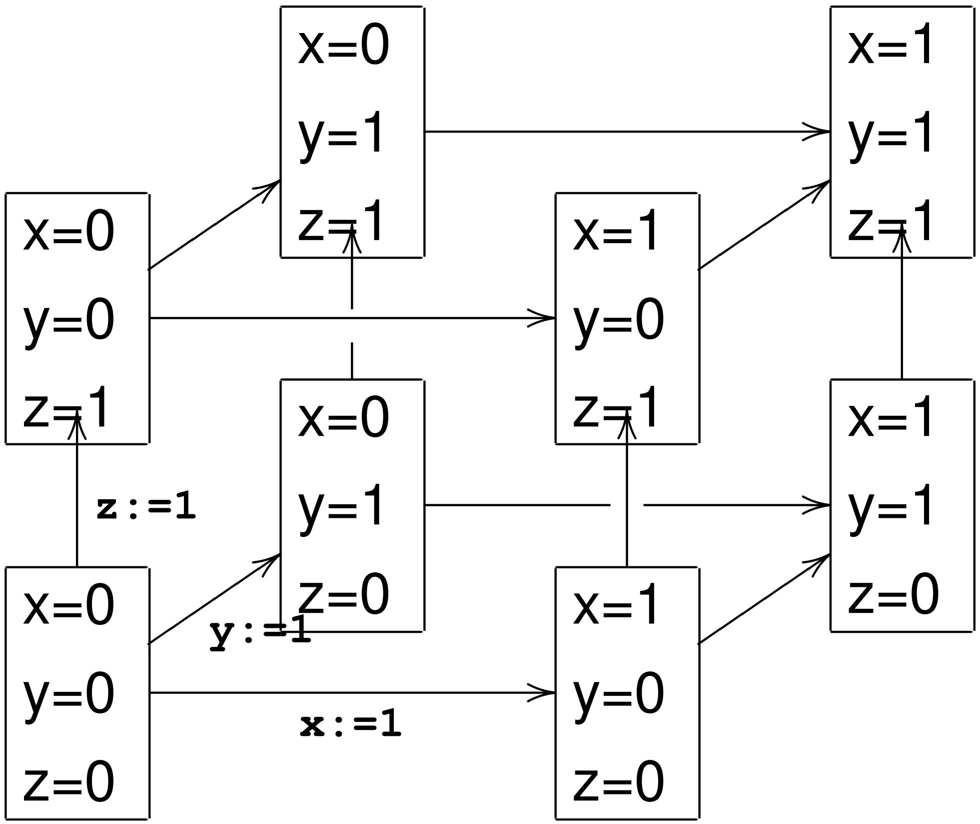
State: a set of pairs of variables and values $\boxed{\begin{array}{l} \text{x=0} \\ \text{y=0} \\ \text{z=0} \end{array}}$

Program: labelled state transition system

$$\boxed{\begin{array}{l} \text{x=0} \\ \text{y=0} \\ \text{z=0} \end{array}} \xrightarrow{\texttt{x:=1}} \boxed{\begin{array}{l} \text{x=1} \\ \text{y=0} \\ \text{z=0} \end{array}} \xrightarrow{\texttt{y:=1}} \boxed{\begin{array}{l} \text{x=1} \\ \text{y=1} \\ \text{z=0} \end{array}} \xrightarrow{\texttt{z:=1}} \boxed{\begin{array}{l} \text{x=1} \\ \text{y=1} \\ \text{z=1} \end{array}}$$

Problem of Model Checking (Parallel Programs): State Explosion

The number of state to be explored increases dramatically (especially in concurrent/parallel programs).



Common Solution: Abstract a Target Program

More concretely:

- keep parts (of a program) that may contain bugs
- remove irrelevant parts (of a program)

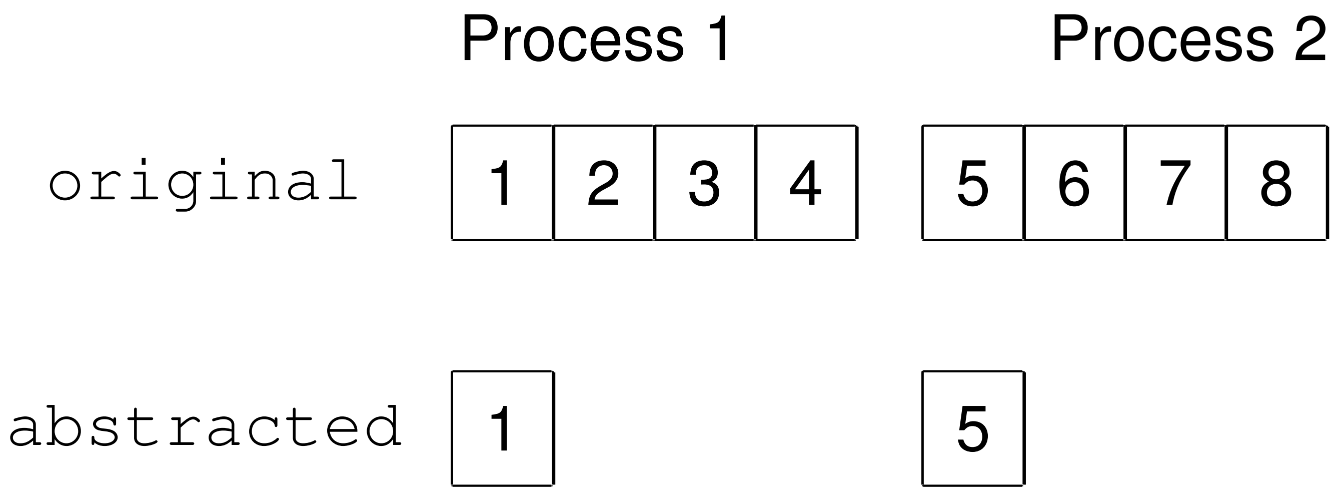
Abstractions introduced in this work

- Shorten lengths of arrays
- Check arrays (in a program) separately

Abstraction 1: Shorten Length of an Array

Observation: bugs occur when accessing boundaries of arrays.

```
!$xmp shadow (0:1) :: a
```



The number of states becomes invariant to w.r.t. the length of an array.

Abstraction 2: Check arrays separately

Observation: arrays are independent of each other w.r.t. missing and redundant directives.

```
!$xmp nodes p(10)
!$xmp shadow (0:1) :: a
!$xmp shadow (0:2) :: b
```

The number of `a`'s boundaries: 10

The number of `b`'s boundaries: 20

Check `a`'s boundaries: 2^{10} states

Check `b`'s boundaries: 2^{20} states

simulaneously: $2^{10} \times 2^{20} = 2^{30}$ states

individually: $2^{10} + 2^{20} \risingdotseq 2^{20}$ states

How to Implement the Abstractions

Updating the source code of programs is tedious, error-prone, and non-productive.

Our approach: design a language for implementing abstraction[1].

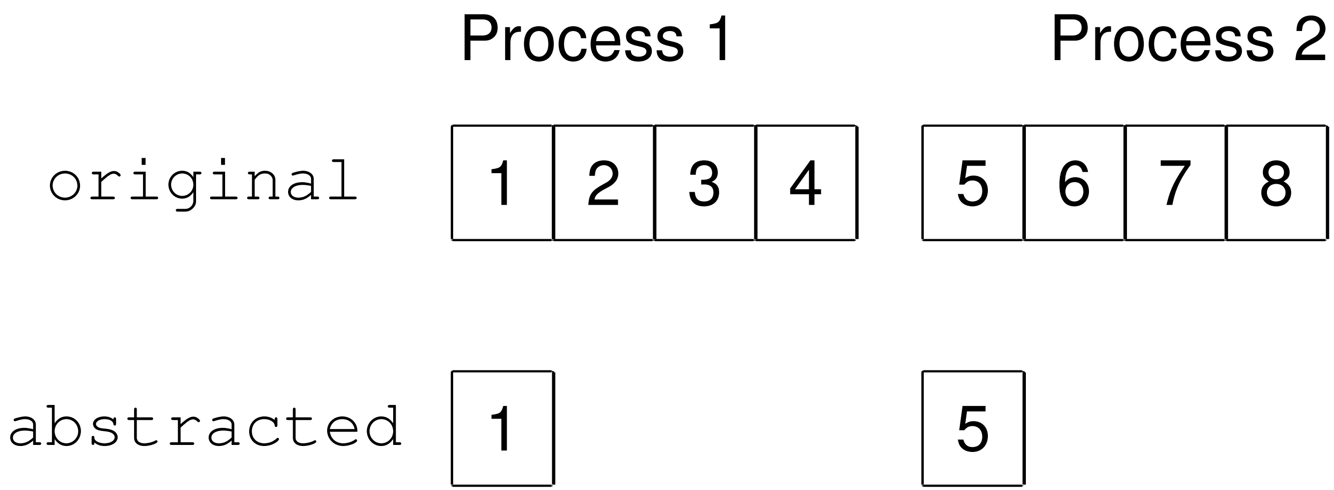
> In our language, we can give abstractions without touching source codes.

[1] Tatsuya Abe, Toshiyuki Maeda, and Mitsuhisa Sato. Model Checking with User-Definable Abstraction for Partitioned Global Address Space Languages. In Proc. of PGAS'12.

Abstraction 1: Shorten Length of an Array

Observation: bugs occur when accessing boundaries of arrays.

```
!$xmp shadow (0:1) :: a
```



The number of states becomes invariant to w.r.t. the length of an array.

Abstraction 1: Shorten Length of an Array

Observation: bugs occur when accessing boundaries of arrays.

```
shrink a sz = shrinker a sz
 where shrinker a sz (Base t a' i) | a == a'
                = [Base (shorten sz t) a i]
       shrinker _ _ d = [d]
       shorten sz (Array t _)
                      = Array t sz
       shorten _ t = t
```

abstracted 1 5

The number of states becomes invariant to w.r.t. the length of an array.

Abstraction 2: Check arrays separately

Observation: arrays are independent of each other w.r.t. missing and redundant directives.

```
!$xmp nodes p(10)
!$xmp shadow (0:1) :: a
!$xmp shadow (0:2) :: b
```

The number of `a`'s boundaries: 10

The number of `b`'s boundaries: 20

Check `a`'s boundaries: 2^{10} states

Check `b`'s boundaries: 2^{20} states

simulaneously: $2^{10} \times 2^{20} = 2^{30}$ states

individually: $2^{10} + 2^{20} \doteqdot 2^{20}$ states

Abstraction 2: Check arrays separately

Observation: arrays are independent of each other w.r.t. missing and redundant directives.

```
!$xmp nodes p(10)
!$xmp shadow (0:1) :: a
!$xmp shadow (0:2) :: b
```

```
slice a = slicer a
 where slicer a (Shadow a' ws) | a == a'
                  = [Shadow a' ws]
       slicer _ (Shadow _ _) = []
       slicer _ d = [d]
```

The

The

Che

Check b's boundaries: 2^{20} states

simulaneously: $2^{10} \times 2^{20} = 2^{30}$ states

individually: $2^{10} + 2^{20} \fallingdotseq 2^{20}$ states

Experimental Results

Target Programs

- Himeno Benchmark (`jacobi`)
- Laplace Equation Solver
- SCALE-LES in XcalableMP

  A library for the simulation of various weather and climate models of the earth and planets.

	lines	arrays	max shadow
Himeno benchmark	65	1	1
Laplace solver	80	1	2
SCALE-LES	1442	13	2

SCALE-LES in XcalableMP

We found 4 errors.

```
$ diff -u scale_unfixed.f90 scale_fixed.f90
--- scale_unfixed.f90   2013-05-20 11:59:16.288925500 +0900
+++ scale_fixed.f90     2013-05-20 11:59:18.397046100 +0900
@@ -353,7 +353,7 @@
 !   end do

   ! memory copy
-!$xmp reflect (dens, pott, momx, momy, momx) ← mis-spelling
+!$xmp reflect (dens, pott, momx, momy, momz)
 ! call copyBoundary (dens)
 ! call copyBoundary (pott)
 ! call copyBoundary (momx)
@@ -1102,6 +1102,7 @@
```

```
        !!! z-direction momentum equation !!!

+!$xmp reflect (work_w2, work_w4) ← missing
 !$xmp loop (ix,jy) on t(ix,jy)
        do jy = JS, JE
           do ix = IS, IE
@@ -1119,9 +1120,7 @@
           end do
        end do

-!$xmp reflect (work_l2) ← redundant
 ! call copyBoundary (work_w2, dim=1)
-!$xmp reflect (work_l4) ← redundant
 ! call copyBoundary (work_w4, dim=2)
 !$xmp loop (ix,jy) on t(ix,jy)
```

SCALE-LES in XcalableMP (Retry)

After fixing the found 4 bugs, we tried to check whether the bugs were (surely) fixed.

Total time: 373 seconds.

21 GiB memory is occupied.

CPU	Intel Xeon X5650 2.67GHz
Memory	24 GB

Summary

- We proposed and implemented model checking of stencil computation written in a PGAS language XcalableMP.
- We implemented abstractions for avoiding the state explosion by utilizing our previous work [PGAS2012].
- We successfully found 4 errors in a real application program (SCALE-LES).

Related Work

Abstraction for program verification for PGAS languages:

MPI-SPIN: S. Siegel. MPI-Spin to Model Check MPI Programs with Nonblocking Communication. Recent Advances in Parallel Virtual Machine and Message Passing Interface, 2006.

UPC-SPIN: A. Ebnenasir. UPC-SPIN: A Framework for the Model Checking of UPC Programs. In Proc. of PGAS'11.

X10X: M. Gligoric et al. X10X: Model Checking a New Programming Language with an "Old" Model Checker. In Proc. of ICST, pages 11–20, 2012.

CAF-SPIN: T. Abe et al. Model Checking with User-Definable Abstraction for Partitioned Global Address Space Languages. In Proc. of PGAS'12.

All the above previous works do not provide any useful feature especially for handling stencil computation.

Future Work

1. Apply our approach to other PGAS languages.

2. Try other computation patterns and verification properties.

3. Extend our approach to support relaxed memory consistency models.
The same program may perform different behaviors on different memory models.